R E P U B L I Q U E F R A N Ç A PCT/FR2004/003275

INPI
INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE

REC'D 25 FEB 2005
WIPO PCT

BREVET D'INVENTION

CERTIFICAT D'UTILITÉ - CERTIFICAT D'ADDITION

COPIE OFFICIELLE

Le Directeur général de l'Institut national de la propriété industrielle certifie que le document ci-annexé est la copie certifiée conforme d'une demande de titre de propriété industrielle déposée à l'Institut.

Fait à Paris, le 22 DEC. 2004

DOCUMENT DE PRIORITÉ

PRÉSENTÉ OU TRANSMIS CONFORMÉMENT À LA RÈGLE 17.1.a) OU b)

Pour le Directeur général de l'Institut national de la propriété industrielle
Le Chef du Département des brevets

Martine PLANCHE

INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE
SIEGE
26 bis, rue de Saint-Petersbourg
75800 PARIS cedex 08
Téléphone : 33 (0)1 53 04 53 04
Télécopie : 33 (0)1 53 04 45 23
www.inpi.fr

ETABLISSEMENT PUBLIC NATIONAL CREE PAR LA LOI N° 51-444 DU 19 AVRIL 1951

1er dépôt

INPI
INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE
26 bis, rue de Saint Pétersbourg
75800 Paris Cedex 08
Téléphone : 33 (1) 53 04 53 04 Télécopie : 33 (1) 42 94 86 54

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI

N° 11354*03

REQUÊTE EN DÉLIVRANCE
page 1/2

BR1

Cet imprimé est à remplir lisiblement à l'encre noire DB 540 @ W / 210502

Réservé à l'INPI	
REMISE DES PIÈCES DATE	30 DEC 2003
LIEU	75 INPI PARIS F
N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI	03 15545
DATE DE DÉPÔT ATTRIBUÉE PAR L'INPI	30 DEC. 2003

1 NOM ET ADRESSE DU DEMANDEUR OU DU MANDATAIRE À QUI LA CORRESPONDANCE DOIT ÊTRE ADRESSÉE

CABINET MOUTARD
B.P. 513
78005 VERSAILLES CEDEX

Vos références pour ce dossier *(facultatif)* TRUSB0023

Confirmation d'un dépôt par télécopie — [X] N° attribué par l'INPI à la télécopie 0315545

2 NATURE DE LA DEMANDE — Cochez l'une des 4 cases suivantes

Demande de brevet	[X]	
Demande de certificat d'utilité	[ ]	
Demande divisionnaire	[ ]	
Demande de brevet initiale	N°	Date
ou demande de certificat d'utilité initiale	N°	Date
Transformation d'une demande de brevet européen *Demande de brevet initiale*	[ ] N°	Date

3 TITRE DE L'INVENTION (200 caractères ou espaces maximum)

CONTROLE D'ACCES AUX DONNEES PAR VERIFICATION DYNAMIQUE DES REFERENCES LICITES.

4 DÉCLARATION DE PRIORITÉ OU REQUÊTE DU BÉNÉFICE DE LA DATE DE DÉPÔT D'UNE DEMANDE ANTÉRIEURE FRANÇAISE

Pays ou organisation
Date N°
Pays ou organisation
Date N°
Pays ou organisation
Date N°

[ ] **S'il y a d'autres priorités, cochez la case et utilisez l'imprimé «Suite»**

5 DEMANDEUR (Cochez l'une des 2 cases) — [X] **Personne morale** [ ] **Personne physique**

Nom ou dénomination sociale		TRUSTED LOGIC
Prénoms		
Forme juridique		société anonyme
N° SIREN		4 2 1 4 8 3 4 1 3
Code APE-NAF		7 2 1 Z
Domicile ou siège	Rue	5, rue du Bailliage
	Code postal et ville	7 8 0 0 0 VERSAILLES
	Pays	France
Nationalité		française
N° de téléphone *(facultatif)*		N° de télécopie *(facultatif)*
Adresse électronique *(facultatif)*		

[ ] **S'il y a plus d'un demandeur, cochez la case et utilisez l'imprimé «Suite»**

Remplir impérativement la 2ème page

1er dépôt

INPI
INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

REQUÊTE EN DÉLIVRANCE
page 2/2

BR2

Réservé à l'INPI

REMISE DES PIÈCES DATE	30 DEC 2003
LIEU	75 INPI PARIS F
N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI	03 15545

DB 540 W / 210502

6 MANDATAIRE *(s'il y a lieu)*		
Nom		de Saint Palais
Prénom		Arnaud
Cabinet ou Société		CABINET MOUTARD
N °de pouvoir permanent et/ou de lien contractuel		
Adresse	Rue	35, rue de la Paroisse
	Code postal et ville	78000 VERSAILLES
	Pays	France
N° de téléphone *(facultatif)*		01 30 83 79 79
N° de télécopie *(facultatif)*		01 30 83 79 78
Adresse électronique *(facultatif)*		asp@moutard.fr
7 INVENTEUR (S)		**Les inventeurs sont nécessairement des personnes physiques**
Les demandeurs et les inventeurs sont les mêmes personnes		☐ Oui ☒ Non : **Dans ce cas remplir le formulaire de Désignation d'inventeur(s)**
8 RAPPORT DE RECHERCHE		**Uniquement pour une demande de brevet (y compris division et transformation)**
Établissement immédiat ou établissement différé		☒ ☐
Paiement échelonné de la redevance *(en deux versements)*		**Uniquement pour les personnes physiques effectuant elles-mêmes leur propre dépôt** ☐ Oui ☐ Non
9 RÉDUCTION DU TAUX DES REDEVANCES		**Uniquement pour les personnes physiques** ☐ Requise pour la première fois pour cette invention *(joindre un avis de non-imposition)* ☐ Obtenue antérieurement à ce dépôt pour cette invention *(joindre une copie de la décision d'admission à l'assistance gratuite ou indiquer sa référence)* : **AG** ☐☐☐☐☐
10 SÉQUENCES DE NUCLEOTIDES ET/OU D'ACIDES AMINÉS		☐ Cochez la case si la description contient une liste de séquences
Le support électronique de données est joint		☐
La déclaration de conformité de la liste de séquences sur support papier avec le support électronique de données est jointe		☐
Si vous avez utilisé l'imprimé «Suite», indiquez le nombre de pages jointes		

11 SIGNATURE DU DEMANDEUR OU DU MANDATAIRE (Nom et qualité du signataire)	**VISA DE LA PRÉFECTURE OU DE L'INPI**
A. de Saint Palais - No 94-0306	

La loi n°78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés s'applique aux réponses faites à ce formulaire. Elle garantit un droit d'accès et de rectification pour les données vous concernant auprès de l'INPI.

La présente invention concerne un procédé pour le contrôle d'accès à des données manipulées par des références dans un système informatique sécurisé.

On sait qu'une des caractéristiques principales des systèmes informatiques sécurisés, qu'ils soient distribués ou non, est le contrôle d'accès aux ressources logicielles du système, et notamment aux programmes et aux données. En particulier, lorsque plusieurs programmes s'exécutent simultanément (ou alternativement) dans le système, on veut s'assurer que l'exécution de l'un n'affecte pas l'exécution des autres, ni celle du système : ils sont isolés.

Plus généralement, on peut laisser certains programmes interagir entre eux, mais uniquement dans le cadre d'une politique stricte de partage contrôlé des données. On se protège ainsi non seulement de la propagation d'erreurs involontaires de programmation, mais aussi et surtout d'actions malveillantes (aussi appelés « attaques ») qui visent à altérer le bon fonctionnement du système et des programmes ou à dévoiler des informations confidentielles.

Par programme, l'on entend ici non seulement le code exécutable, c'est-à-dire une suite d'instructions, mais aussi le processus (ou tâche), c'est-à-dire le code en cours d'exécution, avec son environnement spécifique constitué des données qui lui sont propres ainsi que des ressources qui lui sont attribuées. Par données l'on entend aussi bien des valeurs manipulées par un programme

que les zones de mémoire où sont rangées des valeurs. Selon les systèmes (systèmes d'exploitation, environnements d'exécution, machines virtuelles, etc.), les données appartiennent au programme qui les a créés ou, plus généralement, à un groupe de programmes qui disposent de droits d'accès sur ces données. Ces droits peuvent être donnés à d'autres programmes pour des opérations particulières choisies; de telles données sont dites partageables.

Par exemple, dans le langage « Java Card » (marque déposée de Sun Microsystems), les programmes sont organisés en « packages » (paquetages) à l'intérieur desquels le partage de données (objets et tableaux) est libre. En revanche, l'accès à des données qui appartiennent à un autre « package » est limité par deux dispositifs : un mécanisme de demande d'accès et un mécanisme de pare-feu (« firewall »). En effet, pour accéder à une donnée dont on n'est pas propriétaire, il faut en faire la demande au « package » qui en est propriétaire, demande qu'il peut accepter ou refuser. Par ailleurs, le pare-feu filtre toutes les opérations que l'on peut faire sur une donnée, quel que soit le moyen par lequel on se l'est procurée. En particulier, toute opération de lecture ou d'écriture sur un objet d'un autre « package » est interdite, sauf pour appeler une méthode (routine de programme) explicitement déclarée par ce « package » comme partageable. Il existe également des objets du système (c'est-à-dire de l'environnement d'exécution « Java Card Runtime Environment », ou JCRE) qui sont accessibles (sans droits d'accès particuliers) par tout programme.

Les données d'un programme, et en particulier les données complexes (structures, objets, tableaux, etc.) sont généralement identifiées par une référence. C'est par l'intermédiaire d'une référence que la donnée associée, ainsi que les composants de cette donnée (champs de structures et d'objets, éléments de tableaux, méthodes, etc.), peuvent être manipulés, c'est-à-dire lus, écrits, appelés. La référence elle-même peut être mémorisée, reçue et transmise.

Une référence est souvent un pointeur ou un « handle ». Un pointeur est l'adresse où se trouve rangée une donnée dans la mémoire. Un « handle » est un index dans une table de pointeurs (ou plus généralement dans une table de références). Les valeurs des pointeurs et « handles » comportent aussi parfois des bits spécifiques qui donnent des informations sur la donnée (par exemple sur la zone mémoire référencée ou sur l'information s'y trouvant) ou, dans le cas de « handles », sur la table associée.

Trois attributs majeurs concernent la « correction » des références :

- Une référence peut être valide ou invalide. Une référence valide est une référence associée effectivement à une donnée d'un programme. Une référence invalide est une valeur stockée ou exploitée en tant que référence mais qui n'est pas associée à une donnée. Par exemple, dans le langage C, un pointeur vers une structure de donnée est valide ; en revanche, un pointeur vers l'élément d'un tableau à l'indice –1 est invalide. La validité d'une référence est intrinsèque ; elle ne dépend pas l'agent (programme, processus, tâche, etc.) qui la manipule.
- Une référence peut aussi être licite ou illicite, pour un agent donné du système. Une référence licite est une référence qui a été obtenue par des moyens licites. Une référence illicite est une référence n'a pas été obtenue par des moyens licites. La définition effective de ce qu'est une référence licite ou illicite dépend du système, du langage de programmation et éventuellement du contexte. Par exemple, en langage C, l'arithmétique de pointeur (fabrication d'un pointeur par addition d'un pointeur et d'un entier) est licite ; elle est revanche illicite en « Java ». Une référence n'est pas intrinsèquement licite ou illicite : c'est une propriété spécifique à un agent, qui dépend de la manière dont la référence a été obtenue par cet agent.

- Une référence peut également être « déréférençable » ou « indéréférençable » par un agent donné. Une référence déréférençable par un agent est une référence vers une donnée pour laquelle l'agent a des droits d'accès. Une référence indéréférençable par un agent est une référence vers une donnée pour laquelle l'agent n'a pas de droits d'accès. Par exemple, en « Java Card », une applet qui détient une référence vers un objet peut accéder à un champ de cet objet (ou élément, dans le cas où l'objet est un tableau) à la condition que l'objet appartienne au même package (paquetage) que cette applet. En revanche, si l'objet appartient à un autre package (différent également du JCRE), toute tentative d'accès est stoppée par un mécanisme de pare-feu et résulte en une levée d'exception (de la classe « SecurityException »). Une référence n'est donc pas intrinsèquement déréférençable ou indéréférençable : c'est une propriété spécifique à l'agent qui détient la référence et aux droits d'accès qu'il détient ou non sur la donnée référencée.

Il faut noter que ce sont là trois notions indépendantes.

Ainsi, une référence peut être invalide et licite ; c'est par exemple le cas dans un langage comme C quand on accède aux éléments d'un tableau par arithmétique de pointeur et lorsque l'on dépasse les limites du tableau. Une référence peut aussi être valide et illicite : c'est par exemple le cas de références fabriquées à partir de références connues, dans le cadre d'une attaque, pour accéder à des données protégées auxquelles on ne devrait pas avoir accès. Enfin, une référence peut également être à la fois invalide et illicite ; c'est par exemple le cas de références fabriquées dans le cadre d'une attaque pour écraser les données du programme par balayage systématique et « aveugle » de la mémoire.

Par ailleurs, une référence peut être déréférençable ou non, qu'elle soit valide ou invalide, licite ou illicite. Ainsi, l'accès à une zone mémoire protégée, que la référence soit valide ou non, peut être contrôlé par des droits. D'autre part, on peut obtenir par des moyens illicites une référence vers une donnée sur laquelle on a des droits d'accès ; on peut par exemple la fabriquer de toute pièce au lieu de la demander au système. Inversement, on peut obtenir par des moyens licites une référence vers une donnée sur laquelle on n'a pas des droits d'accès, par exemple pour transmettre cette référence à un autre agent qui, lui, a les droits d'accès appropriés.

L'invention a plus particulièrement pour objet le contrôle des références illicites, les questions de validité et de « déréférençabilité » pouvant être traitées par ailleurs à l'aide de mécanismes qui sont propres à ces notions.

De manière générale, l'accès à une donnée structurée se fait en deux temps. Il faut tout d'abord obtenir une référence sur la donnée. On peut ensuite opérer sur la donnée via sa référence, c'est-à-dire lire ou écrire les composants (champs d'objet ou de structure, éléments de tableau, etc.) de la donnée référencée, ou appeler une de ses routines.

Il y a trois moyens principaux d'obtenir des références :

- Un moyen licite de se procurer une référence est de l'obtenir du système ou d'un autre programme. Par exemple, en « Java » et en « Java Card », les références sont créées par le système lors de l'allocation de nouvelles zones de données en mémoire. Ces références peuvent être fournies et obtenues par un des moyens suivants : lecture et écriture de champs publics d'objets ou de classes, arguments d'appels et valeur de retour de méthodes, levée et rattrapage d'exceptions. En « Java Card », l'accès à un champ public d'un objet appartenant à un autre « package » est toutefois interdit

par le pare-feu. Par ailleurs, une référence à une donnée partageable appartenant au système ou à un autre « package » peut être demandée auprès du système.

- On peut aussi fabriquer une référence à partir d'une autre. Par exemple, dans le langage C, additionner ou soustraire un entier à un pointeur fabrique un nouveau pointeur. Le pointeur résultant est licite car l'arithmétique de pointeurs est licite. Mais il est valide ou non suivant la donnée référencée par le pointeur initial (pointeur à l'intérieur d'un tableau ou d'une structure). Dans d'autres langages que C, le pointeur résultant peut ne pas être licite. C'est le cas pour les langages « Java » et « Java Card », que ce soit au niveau du code source ou du code objet destiné à l'exécution dans une machine virtuelle.
- Enfin, on peut aussi fabriquer une référence de toute pièce. Un simple entier peut ainsi être considéré comme un pointeur ou un « handle » après attribution d'un type référence. La référence résultante peut être valide ou invalide. Elle est en revanche généralement considérée illicite, comme par exemple en « Java » et « Java Card ». Une telle référence peut toutefois être considérée comme licite dans des contextes et pour des langages particuliers. C'est le cas en langage C quand on accède à des ports d'entrée-sortie installés à des adresses mémoires déterminées.

Aux deux temps de l'accès à la donnée (obtenir une référence, puis opérer sur la donnée associée via la référence) correspondent deux mesures couramment employées pour le contrôle d'accès :

- D'une part, il faut empêcher la contrefaçon de références. Autrement dit, un programme ne doit pas pouvoir fabriquer une référence et prétendre l'avoir obtenue par des moyens licites. Comme indiqué ci-dessus, le sens de « moyen licite » varie selon les systèmes, les contextes, et les langages de programmation utilisés. Par exemple, à

la différence du langage C qui permet d'effectuer des opérations arithmétiques sur les pointeurs, les langages « Java » et « Java Card » ont un système de typage des données qui ne permet pas d'effectuer des opérations sur des références ni d'en fabriquer de nouvelles de toute pièce. Les seules références dont un programme peut légitimement disposer en « Java » ou « Java Card » sont celles que lui a fourni le système (y compris la machine virtuelle) ou un autre programme, et qu'il a éventuellement mémorisées. La contrefaçon de références est donc fonctionnellement impossible avec des programmes bien typés. La contrefaçon est toutefois possible au niveau d'une machine virtuelle « Java » ou « Java Card » si elle n'effectue pas de vérification de type, qu'elle soit statique ou dynamique (voir ci-dessous). Même en cas de vérification, il est toujours possible de créer des références illicites par injection de fautes au niveau matériel (bombardement électronique, variations dans l'alimentation électrique, etc.), par exemple dans le cadre d'une attaque contre une carte à puce.

- D'autre part, le système doit filtrer toute opération sur les références. Autrement dit, le programme ne doit pas pouvoir effectuer directement une opération de lecture ou d'écriture sur des données référencées ; il doit passer par l'intermédiaire du système, qui peut accepter ou refuser l'opération, en fonction du programme demandeur et des données référencées (notamment de leur propriétaire). Par exemple, à la différence des instructions du langage machine, qui permettent un accès direct et immédiat à toute donnée référencée, les instructions de la machine virtuelle « Java Card » (JCVM) vérifient systématiquement des conditions de l'accès aux données avant de l'accepter ou de le refuser ; c'est le mécanisme de pare-feu du JCRE. Ce contrôle inclut également l'accès aux éléments d'un tableau : le système vérifie que l'indice des éléments

auxquels on accède reste dans les limites licites, étant donnée la taille du tableau.

Aucune de ces deux mesures, prise individuellement, ne suffit en général à sécuriser un système ; la sécurité repose sur leur combinaison. En fait, les types de protection offerts par chacune de ces deux mesures se recouvrent partiellement et se complètent :

- Ainsi, empêcher la contrefaçon de références garantit qu'un programme ne manipule que des références licites. Cependant, l'accès à toutes les données correspondant à des références licites n'est pas pour autant autorisé ; la présence additionnelle d'un pare-feu permet de restreindre les opérations qu'un programme peut effectuer sur des références, qu'elles aient été obtenues par des moyens licites ou non.
- Toutefois, un pare-feu qui contrôle toutes les opérations sur les références ne suffit pas à garantir la sécurité d'une politique d'accès aux données : il peut être trompé par des références contrefaites. C'est par exemple possible si l'implémentation représente les références par des pointeurs directs vers des blocs mémoire. En effet, dans ce modèle, les descripteurs de données (c'est-à-dire les informations de propriétaire, de classe, de type et de taille de tableau, utilisées lors de la vérification du droit d'accès) sont stockés en mémoire à des déplacements fixes par rapport aux pointeurs. Il suffit à un programme malintentionné de contrefaire un pointeur au milieu d'un bloc de données, par exemple au milieu d'un tableau d'octets convenablement rempli, pour briser l'intégrité de la mémoire : une telle attaque permet en effet de faire croire au système que la donnée associée au pointeur non seulement appartient au programme et est donc accessible, mais aussi que la zone mémoire de cette donnée est d'une étendue arbitrairement grande, ce qui permet de lire ou d'écrire une cellule mémoire quelconque. Ainsi, le contrôle des opérations

sur les références n'interdit pas la fabrication et l'usage d'une référence qui n'a pas été obtenue par des moyens licites ; un accès illicite à une donnée est donc possible malgré le pare-feu.

Le problème de l'interdiction de la contrefaçon de références est généralement résolu de deux manières : par une vérification statique reposant sur une analyse de programme ou par une vérification dynamique des types de valeurs :

- La vérification statique par analyse de programme contrôle statiquement (une fois pour toute, avant toute exécution) que le programme ne peut pas fabriquer lui-même de références sur des données. Autrement dit, une référence ne peut pas être le résultat d'un calcul arbitraire du programme. Dans le cas d'un langage comme « Java » ou « Java Card », cette vérification de type peut s'effectuer aussi bien au niveau d'un programme source (typiquement, à la compilation) qu'au niveau des programmes objets exécutables par une machine virtuelle (typiquement, lors du chargement du programme dans le système ou avant son exécution), à l'aide d'un vérificateur de code octet (« bytecode verifier »).
- La vérification dynamique des types de valeur fait en sorte que les valeurs manipulées par le programme soient marquées par leur type (entier, référence, etc.). On vérifie alors dynamiquement (lors de l'exécution) que seules les valeurs marquées comme étant effectivement des références sont utilisées dans les opérations qui portent sur des références. Le marquage n'est pas nécessairement associé explicitement aux valeurs ; il peut porter sur les zones de stockage des valeurs et être limité à uniquement certaines zones. Par exemple, dans le cas de la machine virtuelle « Java » ou « Java Card », il est suffisant de typer dynamiquement la pile d'opérandes et les variables locales. En revanche, les valeurs stockées dans des tableaux, dans des champs statiques ou dans des champs d'instances

n'ont pas besoin d'être explicitement marquées par leur type, car ce type peut être retrouvé à partir d'autres informations (classe de l'objet, type du tableau) disponibles par ailleurs. Cette approche, dite « pile typée », empêche la contrefaçon de pointeur car elle permet à la machine virtuelle de détecter et d'empêcher toutes les tentatives de conversion d'un entier en référence, ainsi que les opérations arithmétiques sur les références. Une valeur « Java » ou « Java Card » qui a le type référence est ainsi toujours une référence vers un bloc mémoire bien formé du système, contenant des informations correctes de propriétaire, de classe, et de taille.

Ces solutions ne sont pas redondantes ou exclusives ; elles correspondent à des besoins ou à des moyens différents :

- Ainsi, la vérification statique par analyse de programme peut être difficile à mettre en œuvre dans des petits systèmes embarqués tels que les cartes à puce, qui disposent de peu de mémoire et dont la vitesse d'exécution est faible. Cependant, elle a l'avantage de signaler un problème le plus tôt possible. Un programme peut par exemple être rejeté dès son chargement dans le système. (La vérification peut aussi avoir lieu en dehors du système d'exécution et une signature infalsifiable être apposée sur les programmes valides.)
- Pour sa part, la vérification dynamique des types de valeur a l'inconvénient de consommer un peu de mémoire (marque de type ajoutée aux valeurs ou à certaines zones de stockage) et surtout d'être coûteuse en temps d'exécution (affectation et contrôle des marques). Cependant, elle a l'avantage d'être robuste : si un programme a réussi à être chargé dans le système, même par des moyens détournés, il ne pourra de toute façon pas effectuer d'opérations illicites. C'est aussi une garantie supplémentaire contre des attaques matérielles (par opposition aux attaques logicielles). Par exemple, même confinée dans une enceinte sécurisée, comme celle

d'une carte à puce, l'exécution des programmes peut néanmoins être perturbée en agissant sur le matériel (variations de l'alimentation électrique, bombardement par des rayonnements, destructions de portes logiques, etc.). La vérification dynamique est alors un moyen de contrôler régulièrement, à chaque opération sensible, que la politique de contrôle d'accès est bien respectée.

Ces solutions, mises en œuvre partiellement ou en totalité, peuvent être combinées afin de trouver les meilleurs compromis d'efficacité ou fournir la meilleure garantie possible.

Une autre approche, qui ne résout que partiellement le problème lié à la contrefaçon de références, consiste à vérifier que toute valeur utilisée en tant que référence est effectivement une référence connue du système. Cette vérification dynamique de la validité des références n'interdit pas la fabrication de références. Cependant, elle interdit l'usage de références qui en fait n'en sont pas. En d'autres termes, seules les valeurs qui sont effectivement des références peuvent être utilisées en tant que telles par le programme.

La mise en œuvre d'une vérification dynamique de la validité des références dépend fortement de la manière de représenter et de gérer les références dans le système. Par exemple, un gestionnaire de mémoire, auprès de qui l'on peut demander la création d'une nouvelle zone de données ou bien sa libération, sait exactement quelles références ont été créées. Il est donc possible de savoir si un entier correspond ou non à une référence valide existante. Dans le cas où les références sont représentées par des pointeurs, cette opération est toutefois coûteuse car elle peut nécessiter un large parcours des structures de données du gestionnaire de mémoire. En revanche, si les références sont représentées par des « handles », la vérification est bien plus facile et surtout plus rapide : il suffit de vérifier que l'entier est inférieur ou égal à l'indice maximum de la

table de « handles » du gestionnaire de mémoire, et que l'entrée associée dans la table ne correspond pas à des données qui ont été libérées.

Au même titre que les deux vérifications décrites ci-dessus (vérification statique par analyse de programme et vérification dynamique des types de valeur), la vérification dynamique de la validité des références constitue une protection contre les attaques qui fabriquent des pointeurs vers de faux descripteurs de données (attaque décrite ci-dessus). En effet, ces pointeurs contrefaits ne sont pas reconnus par le système comme des références valides existantes et les opérations d'accès sont d'emblée rejetées. Toutefois, ce type de vérification ne garantit pas que le programme s'est procuré par des moyens licites toutes les références qu'il utilise. Un agent peut notamment fabriquer et utiliser une référence valide et déréférençable (référence vers une donnée sur laquelle il a des droits d'accès, par exemple une donnée partageable appartenant à un autre programme) sans qu'elle ait toutefois été obtenue par des moyens licites.

L'invention a donc plus particulièrement pour but de résoudre les problèmes précédemment évoqués.

Elle propose à cet effet une forme de vérification dynamique qui porte sur la contrefaçon de références et qui, d'une certaine manière, complète la vérification dynamique de la validité des références pour couvrir les cas d'utilisation de références illicites.

Conformément à l'invention, cette vérification dynamique des références licites consiste, au cours de l'exécution d'un programme, à :

- mémoriser l'ensemble des références à des données qu'obtient le programme par des moyens licites,
- vérifier, avant toute opération que l'on veut interdire si elle porte sur une valeur qui n'est pas une référence licite, que cette valeur est

parmi les références licites que l'on a mémorisées pour ce programme.

Dans ce procédé, les références pourront consister en des pointeurs ou des « handles ».

Les moyens licites pour un programme d'obtenir des valeurs de référence pourront comprendre au moins l'une des opérations suivantes :

- la lecture d'une variable ou d'une donnée appartenant au système ou à un autre programme,
- l'écriture dans une variable ou donnée dudit programme par le système ou par un autre programme,
- la réception d'arguments à l'appel d'une routine dudit programme par le système ou par un autre programme,
- l'exploitation de la valeur de retour de l'appel par ledit programme d'une routine appartenant au système ou à un autre programme,
- le rattrapage par ledit programme d'une exception levée durant l'exécution d'une routine appartenant au système ou à un autre programme,
- la réception par ledit programme d'une interruption ou d'un signal valué.

Le système pourra disposer d'un mécanisme pour déterminer si une valeur donnée est une référence valide, les références licites mémorisées étant restreintes aux seules références sur des données considérées comme sensibles par le système.

Les susdites vérifications pourront consister à contrôler que les valeurs sont parmi les références licites sensibles que l'on a mémorisées pour ce

programme ou bien sont des références déterminées comme valides et portant sur des données qui ne sont pas sensibles.

Avantageusement, le système pourra disposer d'un mécanisme (dit de pare-feu) qui interdit à certains programmes certaines opérations sur certaines données référencées. Dans ce cas, les données considérées comme sensibles pour le système pourront consister en des données pour lesquelles les opérations ne sont pas interdites par le pare-feu.

De même, le pare-feu pourra interdire à un programme certaines opérations sur des données appartenant à d'autres programmes ou au système, excepté sur celles déclarées comme partageables.

Le système d'exécution de programmes mis en œuvre par le procédé selon l'invention pourra être basé sur une machine virtuelle « Java Card » et dans ce cas :

- le programme exécuté par le système pourra être constitué par l'ensemble du code qui se trouve dans un « package » « Java Card »,
- le mécanisme de pare-feu pourra consister en celui du « Java Runtime Environment » (JCRE),
- les données déclarées comme partageables (et donc sensibles) pourront consister en les objets qui sont des instances de classes qui implémentent l'interface « javacard.framework.Shareable » lorsque ledit « package » appelle une méthode d'un autre « package » ou du système (y compris la méthode « getAppletShareableInterfaceObject » du package « javacard.framework.JCSystem »),
- la lecture d'un champ statique public de type « javacard.framework.Shareable » dans un autre « package » ou dans le système,

- le rattrapage d'un objet instance d'une classe provenant (héritant) de « java.lang.Throwable » et implémentant « javacard.framework. Shareable ».

Dans le procédé précédemment décrit, l'ensemble des références licites (ou licites sensibles) mémorisées pourra être représenté par une table.

Il pourra être vidé, grâce à un ramasse-miettes, des références devenues inactives (c'est-à-dire correspondant à des données effacées du programme ou inutilisables pour un accès futur dans la suite de l'exécution), ce ramasse-miettes pouvant être conservatif.

Les références pourront être représentées dans le système par des « handles » et des tables de pointeurs (ou de références), certaines de ces tables étant éventuellement réservées aux références licites (ou licites sensibles).

Les ensembles de références licites (ou licites sensibles) mémorisées pourront être représentés par des vecteurs (ou matrices) de bits associés à certaines des tables de pointeurs (ou de références) où un bit, à un index donné, représente la présence ou l'absence de la référence correspondante dans lesdits ensembles.

Les vecteurs de bits pourront être éventuellement creux et représentés à l'aide d'une séquence d'indices ou de longueurs correspondant aux étendues de bits positionnés de la même manière (soit à 1, soit à 0).

De même que la vérification dynamique de la validité des références, le mécanisme mis en œuvre par le procédé selon l'invention n'interdit pas la contrefaçon de références. Cependant, il interdit l'accès à des données via des références qui ne peuvent pas être obtenues par des moyens licites. Or c'est ce

contrôle qui seul importe ; que la référence ait été fabriquée de toute pièce en pratique importe peu tant qu'elle est licite.

Par exemple, si le programme construit un entier et tente de l'utiliser comme une référence, soit cet entier ne correspond pas à une référence présente dans l'ensemble des références licites mémorisées pour ce programme et l'opération est rejetée, soit cet entier correspond à une référence déjà présente dans l'ensemble des références licites mémorisées pour ce programme et ne permet donc de faire que des accès licites. L'attaque d'un programme malveillant cherchant à fabriquer une référence vers une donnée partageable devient ainsi inintéressante puisque le programme ne pourra accéder à cette donnée que s'il avait de toute façon pu obtenir cette même référence auparavant, par des moyens licites.

Il est possible de perfectionner ce mécanisme de plusieurs manières :

- d'une part, on peut limiter les références à mémoriser à celles qui importent pour la politique de sécurité du système et limiter le contrôle aux seules opérations que l'on veut voir interdites quand elles portent sur des références illicites ;
- d'autre part, suivant le mode de représentation des références, on peut mettre en œuvre des implémentations plus ou moins efficaces des ensembles de références licites mémorisées.

Des perfectionnements, dont l'objectif est de consommer moins d'espace mémoire et moins de temps d'exécution, sont combinables et seront décrits ci-après à titre d'exemples non limitatifs.

La vérification dynamique des références licites sensibles :

L'accès aux données est parfois cloisonné au sein d'un même programme. Par exemple, en « Java » comme en « Java Card », certains champs d'une classe peuvent être déclarés privés et ainsi n'être accessibles qu'à partir des méthodes de cette classe. Toutefois, ce contrôle d'accessibilité correspond souvent plus à des motivations de génie logiciel qu'à un souci de sécurité. Ce qui importe véritablement en « Java Card » est le cloisonnement entre les données de programmes différents car, en quelque sorte, les programmes « ne se font pas confiance entre eux ». En revanche, à l'intérieur d'un même programme, il importe peu de restreindre les accès possibles. Par exemple, même si une référence vers une donnée du programme est stockée dans un champ privé, et si ce même programme fabrique de manière quelconque un exemplaire de cette référence sans lire ce même champ privé et l'utilise pour accéder à la donnée, la sécurité du programme et de ses données n'est pas mise en danger.

Cette remarque permet de définir un premier perfectionnement du mécanisme de vérification dynamique des références licites en restreignant les références mémorisées et contrôlées. Ce perfectionnement est défini de la manière suivante :

- d'une part, on ne mémorise au cours de l'exécution que l'ensemble des références à des données sensibles qu'obtient un programme par des moyens licites. C'est le contexte ou le type de système qui détermine quelles sont les données que l'on peut considérer comme « sensibles ». Par exemple, dans le cas de « Java Card », on peut ne considérer comme sensibles pour un programme que les données qui appartiennent aux autres programmes. On peut exclure des données considérées comme sensibles les données publiques du système : tableaux globaux (« global arrays ») et objets points d'entrée du JCRE (« JCRE Entry Point Objects »). En effet, bien qu'on ne puisse obtenir des références vers ces données que dans des circonstances particulières (par exemple, retour d'appel de routine), l'accès à ces données est ouvert à tous. En pratique, les données sensibles peuvent

même se réduire aux seules données explicitement déclarées comme partageables, c'est-à-dire aux objets qui sont des instances de classes qui implémentent l'interface « javacard.framework.Shareable », car les autres données sont de toute façon protégées par le mécanisme de pare-feu de la JCVM. Dans ce dernier cas, on mémorise dans l'ensemble des références licites sensibles d'un « package » toutes les références qui apparaissent dans les cas suivants : passage d'arguments de type « Shareable » lorsqu'une méthode du « package » est appelée, valeur de retour de type « Shareable » lorsque le « package » appelle une méthode d'un autre « package » ou du système (y compris la valeur de retour de la méthode « getAppletShareableInterfaceObject » du package « javacard.framework.JCSystem »), lecture d'un champ statique de type « Shareable » dans un autre « package », et rattrapage d'une exception de type « Shareable » ;

- d'autre part, on vérifie, avant toute opération que l'on veut interdire si elle porte sur des valeurs qui ne sont pas des références licites, que la valeur est parmi les références licites sensibles que l'on a mémorisées pour ce programme ou bien est une référence valide sur une donnée qui n'est pas sensible. Il faut noter qu'il est indispensable de disposer également d'un mécanisme de vérification de la validité des références (voir ci-dessus) afin de se prémunir contre des attaques par contrefaçon d'une référence vers un faux descripteur de données. Dans le cas de la JCVM, si l'on ne considère comme sensible que les objets partageables, il suffit que l'instruction « invokeinterface » vérifie, lorsque qu'elle est appliquée à une référence vers un objet appartenant à un autre « package », que cette référence appartient bien à l'ensemble des références licites sensibles associé au « package » appelant.

Cette vérification dynamique des références licites sensibles a l'avantage de consommer moins d'espace mémoire, puisqu'on mémorise moins de références, et moins de temps d'exécution, puisqu'on vérifie moins d'opérations et que le test d'appartenance à l'ensemble des références licites est plus rapide du fait de sa plus petite taille.

Représentation de l'ensemble des références licites :

Par ailleurs, les petits systèmes embarqués tels que les cartes à puce disposent de très peu de mémoire. Il est donc important sur de tels systèmes de pouvoir représenter l'ensemble des références licites (ou licites sensibles) d'un programme de manière compacte tout en permettant une vérification dynamique rapide.

La méthode la plus directe pour représenter l'ensemble des références licites (ou licites sensibles) d'un programme est d'utiliser une table. Lors de l'introduction dans le programme d'une référence par un moyen licite, on l'ajoute dans la table si elle n'y est pas déjà présente. La vérification qu'une référence est licite se fait par examen successif des entrées dans la table.

D'autres manières standards en algorithmique pour représenter des ensembles peuvent aussi être employées : listes, arbres, etc. Certaines représentations des ensembles permettent notamment d'optimiser les opérations d'ajout, de suppression et de test de présence d'un élément dans un ensemble lorsqu'on connaît à l'avance le nombre maximum (ou maximum probable) d'éléments. On peut en effet alors dimensionner l'ensemble selon ce nombre maximum et faire des accès plus ou moins directs aux éléments.

Nettoyage de l'ensemble des références licites :

D'autre part, il faut veiller à la cohérence des références licites (ou licites sensibles) mémorisées en cas de suppression de données.

En effet, en cours d'exécution, des données devenues inutiles ou inaccessibles peuvent être supprimées de la mémoire du système ou d'un programme. Une donnée devient inaccessible dès lors que le programme ne contient plus de référence active sur cette donnée, c'est-à-dire encore susceptible d'être utilisée par le programme dans la suite de son exécution. De telles données peuvent être effacées explicitement, par l'appel à une routine de libération de mémoire, ou automatiquement, par un ramasse-miettes (« garbage collector »).

Si une référence devient inactive dans un programme, l'ensemble des références licites (ou licites sensibles) mémorisées reste compatible avec la sécurité du système : le programme est de toute façon libre d'utiliser ou non les références dont il dispose, et l'on peut contrôler que toutes celles qu'il utilise sont bien licites. Un tel ensemble peut toutefois être nettoyé des références inactives, afin de réduire son encombrement en mémoire. Ce nettoyage devient même impératif si l'allocateur de données peut créer une nouvelle donnée associée à une référence dont la donnée a été supprimée.

Le nettoyage de l'ensemble des références licites (ou licites sensibles) peut être effectué automatiquement, par un ramasse-miettes. Il doit pour cela parcourir toutes les valeurs présentes dans le programme en cours d'exécution afin de déterminer les références encore actives. Toutes les références rencontrées lors de ce parcours sont marquées comme « encore en service » dans l'ensemble des références licites du programme. À la fin de ce parcours, toutes les entrées non marquées peuvent être libérées : elles correspondent à des données auxquelles le programme a pu accéder dans le passé, mais sur lesquelles il n'a pas gardé de référence (ou pas de référence utilisable).

Dans le cas où des données peuvent ainsi être effacées, il n'est pas nécessaire de dimensionner l'ensemble des références licites selon le nombre de données référencées dans le système : on peut le sous-dimensionner et le débarrasser régulièrement, par exemple lorsque l'ensemble est plein, des éléments qui ne sont plus utiles. Ainsi, il suffit de dimensionner l'ensemble des références licites d'un programme au nombre de références simultanément actives lors de son exécution, nombre qui est plus petit que le nombre de données référencées dans le système.

Par ailleurs, dans le cas où l'on ne sait pas décider avec certitude si une valeur représente ou non une référence, ce qui est le cas dans un système d'exécution (y compris une machine virtuelle) qui ne conserve pas toutes les informations de type, on peut avoir recours à un ramasse-miettes dit « conservatif ». Avec un tel ramasse-miettes, toute valeur susceptible d'être une référence (par exemple un entier) est considérée comme telle par sécurité. Ce ramasse-miettes est dit conservatif car les valeurs prises à tort pour des références empêchent le nettoyage de ces références ; en revanche, on est sûr qu'aucune référence ne peut être supprimée tant qu'elle est encore active.

Représentation des références licites par filtrage des tables de « handles » :

Enfin, lorsque les références sont représentées par des « handles », auxquels correspondent des index associés à une ou plusieurs tables de pointeurs (ou de références) gérées par le système, l'ensemble des références licites (ou licites sensibles) d'un programme peut être représenté de manière plus compacte.

On peut employer pour cela des vecteurs de bits de même taille que les tables de pointeurs. Ces vecteurs sont interprétés comme des filtres sur les tables de pointeurs (ou de références) pour indiquer les références considérées comme présentes dans l'ensemble : un bit levé (égal à 1) ou non (égal à 0) à un index donné indique si la référence correspondante doit être considérée ou non dans

l'ensemble des références licites (ou licites sensibles) du programme. L'ajout, la suppression et le test de présence d'un élément dans l'ensemble est extrêmement rapide puisqu'il n'y a qu'un bit à positionner ou tester. En attribuant des numéros aux programmes, on peut aussi regrouper ces vecteurs de bits en une matrice de bits dont une des coordonnées est indexée par le numéro du programme.

S'il y a beaucoup de références dans le système et si très peu sont à considérer comme licites (ou licites sensibles) pour chacun des programmes, cette matrice de bits peut finalement s'avérer moins compacte qu'une représentation par simple table de références explicites. Dans ce cas, on peut essayer d'employer une représentation sous forme de vecteur creux (ou de matrice creuse), par exemple une séquence d'indices ou de longueurs correspondant aux étendues de bits positionnés de la même manière (soit à 1, soit à 0).

On peut aussi exploiter des tables différentes pour mémoriser d'une part les « handles » qui sont des références licites (ou licites sensibles) et d'autres part les autres références. Les vecteurs de bits deviennent ainsi beaucoup moins longs et beaucoup plus denses.

Revendications

1. Procédé de contrôle d'accès à des données manipulées par références dans un système d'exécution de programmes (y compris processus et tâches), caractérisé en ce que, lors de l'exécution d'un programme, il comprend les étapes suivantes :

- la mémorisation par le système de l'ensemble des références qu'obtient le programme par des moyens considérés comme licites ;
- avant toute opération que l'on veut interdire si elle porte sur des valeurs qui ne sont pas des références licites, la vérification par le système que ces valeurs sont parmi les références licites que l'on a mémorisées pour ce programme, et l'acceptation ou le rejet de l'opération en conséquence.

2. Procédé selon la revendication 1, caractérisé en ce que les références sont des pointeurs et/ou des « handles ».

3. Procédé selon la revendication 1, caractérisé en ce que les moyens licites pour un programme d'obtenir des valeurs références comprennent au moins l'une des opérations suivantes :

- la lecture d'une variable ou d'une donnée appartenant au système ou à un autre programme,
- l'écriture dans une variable ou donnée dudit programme par le système ou par un autre programme,
- la réception d'arguments à l'appel d'une routine dudit programme par le système ou par un autre programme,
- l'exploitation de la valeur de retour de l'appel par ledit programme d'une routine appartenant au système ou à un autre programme,
- le rattrapage par ledit programme d'une exception levée durant l'exécution d'une routine appartenant au système ou à un autre programme,
- la réception par ledit programme d'une interruption ou d'un signal valué.

4. Procédé selon la revendication 1,

caractérisé en ce que :

- le système comprend un mécanisme qui détermine si une valeur donnée est une référence valide, et/ou
- les références licites mémorisées sont restreintes aux seules références sur des données considérées comme sensibles pour le système, et/ou
- lesdites vérifications contrôlent que les valeurs sont parmi les références licites sensibles que l'on a mémorisées pour ce programme ou bien sont des références déterminées comme valides et portant sur des données qui ne sont pas sensibles.

5. Procédé selon la revendication 4,

caractérisé en ce que le système comprend un pare-feu qui interdit à certains programmes certaines opérations sur certaines données référencées, les données considérées comme sensibles pour le système étant celles pour lesquelles les opérations ne sont pas interdites par le pare-feu.

6. Procédé selon la revendication 5,

caractérisé en ce que le pare-feu interdit à un programme certaines opérations sur des données appartenant à d'autres programmes, excepté sur celles déclarées comme partageables.

7. Procédé selon la revendication 6,

caractérisé en ce que le système est basé sur une machine virtuelle « Java Card » et en ce que :

- un programme est constitué par l'ensemble du code qui se trouve dans un « package Java Card » ;
- le pare-feu est celui du « Java Card Runtime Environment » (JCRE) ;
- les données déclarées comme partageables (et donc sensibles) sont les objets qui sont instances de classes qui implémentent l'interface « javacard.framework.Shareable » ainsi que, éventuellement, les objets à

usage public du système : tableaux globaux (« global arrays ») et objets points d'entrée du JCRE (« JCRE Entry Point Objects »).

8. Procédé selon la revendication 7,
caractérisé en ce que le système mémorise dans les ensembles de références licites sensibles associés à un « package » toutes les références qui apparaissent dans les cas suivants :

- la réception d'arguments de type « javacard.framework.Shareable » lorsqu'une méthode dudit package est appelée par un autre « package » ou par le système,
- la valeur de retour de type « javacard.framework.Shareable » lorsque ledit « package » appelle une méthode d'un autre « package » ou du système (y compris la méthode « getAppletShareableInterfaceObject » du package « javacard.framework.JCSystem »),
- la lecture d'un champ statique public de type « javacard.framework.Shareable » dans un autre package ou dans le système,
- le rattrapage d'un objet instance d'une classe provenant (héritant) de «java.lang.Throwable» et implémentant «javacard.framework.Shareable».

9. Procédé selon l'une des revendications 1 et 4,
caractérisé en ce que l'ensemble des références licites (ou licites sensibles) mémorisées est représenté par une table.

10. Procédé selon l'une des revendications 1 et 4,
caractérisé en ce que l'ensemble des références licites (ou licites sensibles) mémorisées est vidé, grâce à un ramasse-miettes, éventuellement conservatif, des références devenues inactives.

11. Procédé selon l'une des revendications 1 et 4,

caractérisé en ce que :

- les références sont représentées dans le système par des « handles » et des tables de pointeurs (ou de références),
- certaines desdites tables sont éventuellement réservées aux références licites (ou licites sensibles),
- les ensembles de références licites (ou licites sensibles) mémorisées sont représentés par des vecteurs (ou matrices) de bits associés à certaines des tables de pointeurs (ou de références), où un bit à un index donné représente la présence ou l'absence de la référence correspondante dans lesdits ensembles,
- lesdits vecteurs de bits sont éventuellement creux et représentés à l'aide d'une séquence d'indices ou de longueurs correspondant aux étendues de bits positionnés de la même manière.

reçue le 07/01/04

INPI
INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE

DÉPARTEMENT DES BREVETS

26 bis, rue de Saint Pétersbourg
75800 Paris Cedex 08
Téléphone : 33 (1) 53 04 53 04 Télécopie : 33 (1) 42 94 86 54

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI

cerfa N° 11235*03

DÉSIGNATION D'INVENTEUR(S) Page N° 1../2..

INV

(À fournir dans le cas où les demandeurs et les inventeurs ne sont pas les mêmes personnes)

Cet imprimé est à remplir lisiblement à l'encre noire

DB 113 @ W / 270601

Vos références pour ce dossier *(facultatif)*	TRUSB0023
N° D'ENREGISTREMENT NATIONAL	03 15545 du 30.12.2003

TITRE DE L'INVENTION (200 caractères ou espaces maximum)

CONTROLE D'ACCES AUX DONNEES PAR VERIFICATION DYNAMIQUE DES REFERENCES LICITES.

LE(S) DEMANDEUR(S) :

CABINET MOUTARD - 35, rue de la Paroisse - 78000 VERSAILLES - agissant en qualité de mandataire auprès de :
TRUSTED LOGIC (S.A.)
5, rue du Bailliage
78000 VERSAILLES

DESIGNE(NT) EN TANT QU'INVENTEUR(S) :

1 Nom		LEROY
Prénoms		Xavier
Adresse	Rue	37, boulevard Saint Antoine
	Code postal et ville	|7|8|0|0|0| VERSAILLES
Société d'appartenance *(facultatif)*		
2 Nom		HAMEAU
Prénoms		Patrice
Adresse	Rue	18, rue de Belle-Feuille
	Code postal et ville	|9|2|1|0|0| BOULOGNE BILLANCOURT
Société d'appartenance *(facultatif)*		
3 Nom		REGNAULT
Prénoms		Nicolas
Adresse	Rue	66, rue Bayen
	Code postal et ville	|7|5|0|1|7| PARIS
Société d'appartenance *(facultatif)*		

S'il y a plus de trois inventeurs, utilisez plusieurs formulaires. Indiquez en haut à droite le N° de la page suivi du nombre de pages.

DATE ET SIGNATURE(S)
DU (DES) DEMANDEUR(S)
OU DU MANDATAIRE
(Nom et qualité du signataire)

31 décembre 2003

A. de Saint Palais - No 94-0306

La loi n°78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés s'applique aux réponses faites à ce formulaire.
Elle garantit un droit d'accès et de rectification pour les données vous concernant auprès de l'INPI.

reçue le 07/01/04

INPI
INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI

N° 11235*03

DÉPARTEMENT DES BREVETS
26 bis, rue de Saint Pétersbourg
75800 Paris Cedex 08
Téléphone : 33 (1) 53 04 53 04 Télécopie : 33 (1) 42 94 86 54

DÉSIGNATION D'INVENTEUR(S) Page N° 2../2..

(À fournir dans le cas où les demandeurs et les inventeurs ne sont pas les mêmes personnes)

INV

Cet imprimé est à remplir lisiblement à l'encre noire

DB 113 @ W / 270601

Vos références pour ce dossier *(facultatif)*	TRUSB0023
N° D'ENREGISTREMENT NATIONAL	03 15545 du 30.12.2003

TITRE DE L'INVENTION (200 caractères ou espaces maximum)

CONTROLE D'ACCES AUX DONNEES PAR VERIFICATION DYNAMIQUE DES REFERENCES LICITES.

LE(S) DEMANDEUR(S) :

CABINET MOUTARD - 35, rue de la Paroisse - 78000 VERSAILLES - agissant en qualité de mandataire auprès de :
TRUSTED LOGIC (S.A.)
5, rue du Bailliage
78000 VERSAILLES

DESIGNE(NT) EN TANT QU'INVENTEUR(S) :

1 Nom		MARLET
Prénoms		Renaud
Adresse	Rue	109 Avenue du Maine
	Code postal et ville	7 5 0 1 4 PARIS
Société d'appartenance *(facultatif)*		
2 Nom		
Prénoms		
Adresse	Rue	
	Code postal et ville	
Société d'appartenance *(facultatif)*		
3 Nom		
Prénoms		
Adresse	Rue	
	Code postal et ville	
Société d'appartenance *(facultatif)*		

S'il y a plus de trois inventeurs, utilisez plusieurs formulaires. Indiquez en haut à droite le N° de la page suivi du nombre de pages.

DATE ET SIGNATURE(S)
DU (DES) DEMANDEUR(S)
OU DU MANDATAIRE
(Nom et qualité du signataire)

31 décembre 2003

A. de Saint Palais - No 94-0306

La loi n°78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés s'applique aux réponses faites à ce formulaire.
Elle garantit un droit d'accès et de rectification pour les données vous concernant auprès de l'INPI.

PCT/FR2004/003275